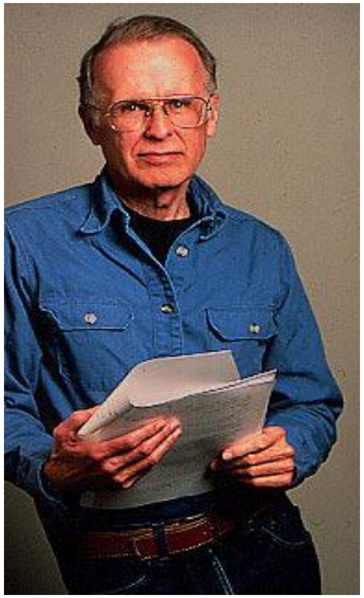
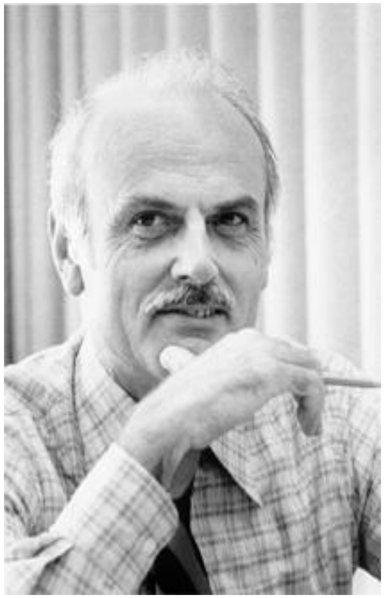
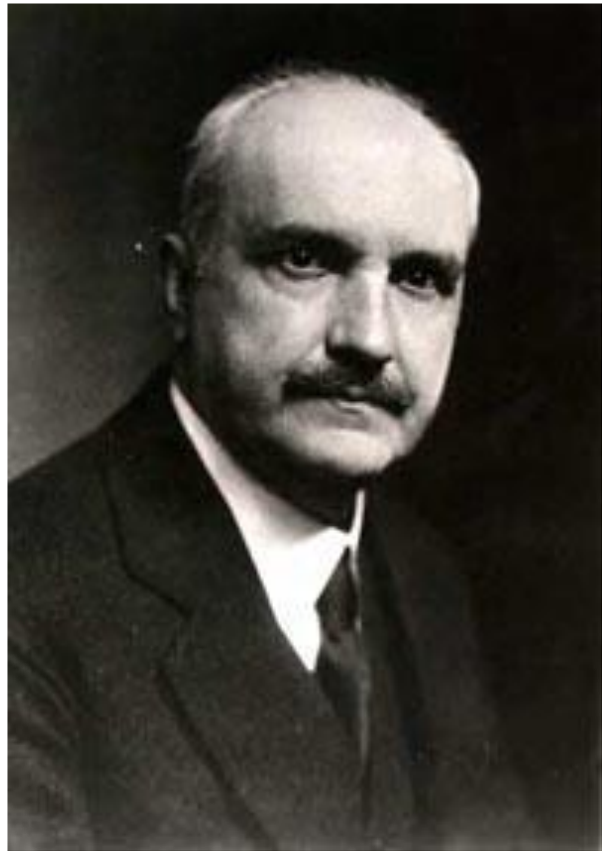
Parallel Programming:

From  to 

Keshav Pingali
University of Texas, Austin

Patron saint of parallel computing



Jorge Agustín Nicolás Ruíz de Santayana y Borrás (1863-1952)

“Those who remember Santayana’s saying are condemned to repeat it”.

“Those who forget the past are condemned to repeat it”.

Lessons from Parallel Computing 1.0

- Successes
  - numerical linear algebra
  - databases
- Successes waiting to happen
  - functional and logic programming
  - parallelizing compilers for FORTRAN/C

What lessons can we learn from these?

Lesson #1

Success requires deep understanding of application areas.

Fallacy:

Compilers, architectures, etc. know nothing about application areas
➔
Compiler writers, architects, etc. need not know anything about application areas

Running benchmarks is not the same as understanding application areas.

- Result:
  - research is not focused on solving particular problems, but on inventing new mechanisms
  - mechanism evaluations use either micro-benchmarks, or large benchmarks that no one understands
  - ➔ most mechanisms are of dubious utility
    - (eg) array dependence tests
  - ➔ no notion of "finishing" problems
    - (eg) loop transformations for locality enhancement

Lesson #1 (contd.)

- ☹ Benchmarks: black/brown boxes
  - I/O behavior + data from probes
  - Provides limited insights
- ☺ Program = Algorithm + Data Structure (Wirth)
  - Need to study algorithms and data structures, not programs
  - Frequent pattern mining
    - What are the common algorithm and data structure patterns?
    - How do we support them efficiently?

Lesson #2

There are two classes of programmers
- Joe programmers: domain experts, not expert parallel programmers
- Steve programmers: domain experts who know a lot about parallel programming

- Numerical linear algebra
  - Goto, Dongarra et al: Steves (BLAS, LAPACK)
  - Matlab users: Joes
- Databases
  - Database implementers: Steves
  - SQL programmers: Joes

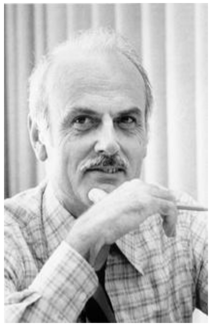
Lesson #2: (contd.)

This strategy requires

- proper division of labor
  - small number of Steves
  - to support a large number of Joes
- careful attention to contract between Joe and Steve
  - contract is much more than an API
  - contract is an *information model* or *ontology*

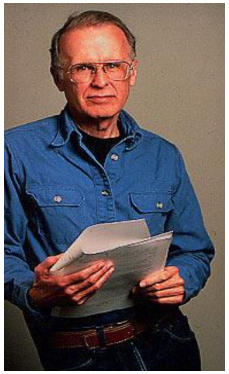
Information model (Wikipedia)

- Formal representation of entities that includes
  - properties of entities
  - relationships between entities
  - operations on entities
  - properties of operations
- Some information models are *computational algebras*
  - (eg) relational algebra in databases
  - but some are not (eg) BLAS interface in dense linear algebra
- Motivation
  - “provide formal description of domain without constraining implementation of description” (Wikipedia)



Information models: databases

- Codd’s 12 rules for relational databases
  - Rule 8: Physical data independence
    - The user should not be aware of where or upon which media data-files are stored.
  - Rule 9: Logical data independence
    - User programs and the user should not be aware of any changes to the structure of the tables such as the addition of additional columns.
  - Rule 11: Distribution independence
    - The RDBMS may be distributed across more than one system and across several networks, but to the end-user, the tables should appear no different than those that are local.



Contrast:general-purpose PL

- No clear delineation of roles between
  - different classes of programmers
  - programmers and compilers
- Permit optimization by programmers and by compilers
  - no distinction between
    - abstraction and implementation: implicit array reshaping in FORTRAN
    - data and meta-data: pointers in C, representation exposure in OO languages

➔ Everyone and every system involved in the programming process is responsible for everything and nothing.

Что делать?*

- Difficult problem
  - we need to support many information models, not just matrices and relations
  - matrices and relations are relatively simple
    - flat data models
    - cf. sparse matrix information model: much harder
- Scientific principle:
  - when faced with a difficult problem, simplify
    - specialization: don't try to solve everything at the same time
    - abstraction: ignore details (friction, center of mass,..)

* "What is to be done?" Lenin (1901)

From Backus to Codd: one strategy

1. Recognize
   - general-purpose, high-level, efficient parallel programming is too ambitious a goal right now
2. Specialize
   - focus on specific domains that need high performance computing and have lots of Joes and few Steves
   - work closely with domain scientists to figure out information models for those domains
   - systems work:
     - language and compiler work:
       - high-level optimization of Joe's programs using information model properties
       - decompose Joe's programs into efficient sequences of information model API
         - this can be quite tricky: dense matrix factorizations
     - compiler/runtime/architecture work:
       - make Steves' job easier
       - focus on particular domains may make job easier
3. Generalize
   - after successes in enough domains

Implications of strategy

- Designing new languages is not the solution by itself.
  - "Languages can be the problem but they are rarely the solution" (advice from David Kuck ca. 1986)
- Dusty deck problem: ignore it for now
  - Solve simpler problems before tackling more difficult ones
- Languages/systems research must be performed and evaluated in context
  - No more "open loop" research
  - Must be willing to recognize/reward domain-specific work
- Specialization of techniques
  - May need analysis and optimization techniques at multiple scales

    (eg.) does empirical optimization need to scale up to large programs?

Patron saint of parallel computing

"Pessimism of the intellect, optimism of the will"

Antonio Gramsci (1891-1937)